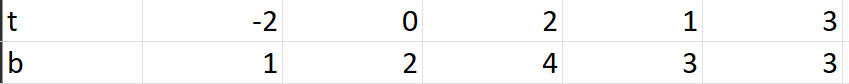
Vladislav Grigorev, CS-03, [v.grigorev@innopolis.university](mailto:v.grigorev@innopolis.university)

Full code: <https://github.com/vladislav77777/AGLA_ass-2/blob/main/src>

// Vladislav Grigorev  
// 31.03.2023  
// v.grigorev@innopolis.university  
/\*  
 \* This is a C++ program that defines several classes: Matrix, squareMatrix, identityMatrix, elimination and permutation Matrices.  
 \* The Matrix class has member functions to perform matrix addition, subtraction, and multiplication.  
 \* Some classes are inherit from others. The task is to compose a least square approximation for a given data set.  
 \* For this purpose we have class «ColumnVector» with necessary fields, methods and necessary operators'  
 \* overloading for summation, multiplication, inputting-outputting and computing the norm.  
 \*/  
#include <iostream>  
#include <vector>  
#include <cmath>  
#include <iomanip>  
#include <cstdlib>  
#include <cstdio>  
  
  
#ifdef WIN32  
#define GNUPLOT\_NAME "C:\\gnuplot\\bin\\gnuplot -persist"  
#endif  
  
int main() {  
#ifdef WIN32  
 FILE \*pipe = \_popen(GNUPLOT\_NAME, "w");  
#endif  
 if (pipe != NULL) {  
 colNUM = 0; // it's pivot  
// ios\_base::sync\_with\_stdio(false);  
// cin.tie(nullptr);  
 int m, n;  
 cin >> m;  
 vector<double> t(m);  
 ColumnVector b((vector<double>(m)));  
 fprintf(pipe, "%s\n",  
 "plot '-' using 1:2 title 'points' with points pointtype 5, '-' using 1:2 title 'LSA' with lines");  
 for (int i = 0; i < m; i++) {  
 cin >> t[i] >> b.values[i];  
 fprintf(pipe, "%f\t%f\n", t[i], b.values[i]);  
 }  
 fprintf(pipe, "%s\n", "e");  
  
 // Construct the matrix A  
 cin >> n;  
 Matrix A(m, n + 1);  
 for (int i = 0; i < m; i++) {  
 for (int j = 0; j < n + 1; j++) {  
 A.matrix[i][j] = pow(t[i], j);  
 }  
 }  
 cout << "A:\n" << A;  
 // Construct the transpose of A  
 Matrix AT(n + 1, m);  
 for (int i = 0; i <= n; i++) {  
 for (int j = 0; j < m; j++) {  
 AT.matrix[i][j] = A.matrix[j][i];  
 }  
 }  
 Matrix AT\_A = AT \* A;  
 cout << "A\_T\*A:\n" << AT\_A;  
  
 identityMatrix I(AT\_A.rows, AT\_A.columns);  
 if (!(isAllZeroes(AT\_A))) // until we don't have all zeroes under main diagonal  
 while (!(isAllZeroes(AT\_A))) {  
 if (isNeedPermutation(AT\_A)) { // if we need swap rows(max by absolute value)  
 permutMatr(AT\_A, I); // create permutation matrix and multiply it by A  
 } else {  
 auto E = new eliminationMatrix(AT\_A.rows, AT\_A.columns);  
 E->eliminate(AT\_A); // if not we are creating elimination matrix  
 AT\_A = (Matrix) \*E \* AT\_A; // and multiply it by A  
 I = (\*E) \* I;  
 }  
 }  
  
 if (!(isAllZeroesBack(AT\_A))) // until we don't have all zeroes above main diagonal  
 while (!(isAllZeroesBack(AT\_A))) {  
 auto E = new eliminationMatrix(AT\_A.rows, AT\_A.columns);  
 E->eliminateBack(AT\_A); // make all zeroes above main diagonal  
 AT\_A = (Matrix) \*E \* AT\_A;  
 I = \*E \* I;  
 }  
  
 for (int i = 0, j = 0; i < AT\_A.rows; i++, j++) { // doing diagonal normalization to obtain identity matrix A  
 for (int k = 0; k < I.columns; k++) {  
 I.matrix[i][k] /= AT\_A.matrix[i][j];  
 }  
 AT\_A.matrix[i][j] /= AT\_A.matrix[i][j];  
 }  
 Matrix AT\_Atr = I;  
 cout << "(A\_T\*A)^-1:\n" << AT\_Atr;  
 Matrix AT\_b = AT \* b;  
 cout << "A\_T\*b:\n" << AT\_b;  
 Matrix ans = (AT\_Atr \* AT\_b);  
 cout << "x~:\n" << ans;  
 const double maxV = 6;  
 const double npoints = 50;  
 const double step = maxV / npoints;  
 for (int i = 0; i < npoints + 1; i++) {  
 double x = -maxV / 2 + i \* step;  
 double y = 0;  
 for (int j = 0; j < ans.matrix.size(); j++) {  
 y += pow(x, j) \* ans.matrix[j][0];  
 }  
 fprintf(pipe, "%f\t%f\n", x, y);  
 }  
  
 fprintf(pipe, "%s\n", "e");  
  
 } else  
 cout << "Could not open pipe" << endl;  
  
 fflush(pipe);  
#ifdef WIN32  
 \_pclose(pipe);  
#endif  
}

Set of points

Approximation

